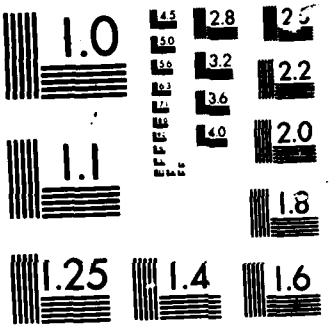
AD-A166 776 INSTRUCTIONAL SUPPORT SOFTWARE(U) MCDONNELL DOUGLAS ASTRONAUTICS CO-ST LOUIS MO MAR 86 AFHRL-TR-85-53 F33615-81-C-0021 1/1

UNCLASSIFIED F/G 9/2 NL



MICROCOPY CHART

AFHRL-TR-85-53

(12)

AIR FORCE

HUMAN RESOURCES

LABORATORY

AD-A166 776

DTIC FILE COPY

INSTRUCTIONAL SUPPORT SOFTWARE SYSTEM

McDonnell Douglas Astronautics Company
P. O. Box 516
St. Louis, Missouri 63166

TRAINING SYSTEMS DIVISION
Lowry Air Force Base, Colorado 80230-5000

March 1986

Final Report for Period July 1981 - September 1985

DTIC
ELECTE
APR 17 1986
S D
E

Approved for public release; distribution unlimited.

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This report has been reviewed and is approved for publication.

ALAN MARSHALL
Contract Monitor

JOSEPH Y. YASUTAKE, Technical Advisor
Training Systems Division

DENNIS W. JARVI, Colonel, USAF
Commander

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

ADA 166 776

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFHRL-TR-85-53
6a. NAME OF PERFORMING ORGANIZATION McDonnell Douglas Astronautics Company 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION Training Systems Division Air Force Human Resources Laboratory
6c. ADDRESS (*City, State, and ZIP Code*) P.O. Box 516 St. Louis, Missouri 63166	7b. ADDRESS (*City, State, and ZIP Code*) Lowry Air Force Base, Colorado 80230-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Human Resources Laboratory 8b. OFFICE SYMBOL (*If applicable*) HQ AFHRL	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-81-C-0021
8c. ADDRESS (*City, State, and ZIP Code*) Brooks Air Force Base, Texas 78235-5601	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62205F; PROJECT NO. 1121; TASK NO. 09; WORK UNIT ACCESSION NO. 07
11. TITLE (*Include Security Classification*) Instructional Support Software System	
12. PERSONAL AUTHOR(S)	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM Jul 81 TO Sep 85	14. DATE OF REPORT (*Year, Month, Day*) March 1986 15. PAGE COUNT 26
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
05	09	

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
Ada; advanced instructional system; computer-assisted instruction; computer-managed instruction; instructional support software; transportable instruction system

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This report describes the development of the Instructional Support System (ISS), a large-scale, computer-based training system that supports both computer-assisted instruction (CAI) and computer-managed instruction (CMI). The ISS is a software package that is written in Ada, designed to be machine independent, and grouped into functional modules so that each module can be executed individually or can be combined as needed to support operational requirements. The ISS is not designed for any particular machine or operating system and, hence, can run on computer systems ranging from small microcomputers on up. The ISS is a Government product available on a no-cost basis to authorized agencies or organizations.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Nancy A. Perrigo, Chief, STINFO Office	22b. TELEPHONE (*Include Area Code*) (512) 536-3877 22c. OFFICE SYMBOL AFHRL/TSR

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

SUMMARY

A team comprised of personnel from AFHRL, McDonnell Douglas Astronautics Company, Denver Research Institute, and Softech determined requirements for and developed the Instructional Support System (ISS). The basis for this development was the Advanced Instructional System (AIS), a computer-based instructional system developed jointly by AFHRL and McDonnell Douglas Astronautics Company.

Early in the project, McDonnell Douglas teamed with Denver Research Institute to determine the functional requirements of the ISS. An examination of key DOD training systems occurred to determine the feasibility of adding certain key features to the ISS at reasonable cost. Also, several key DOD training environments were examined to determine training requirements the ISS would need to address. Existing AIS capabilities as well as inputs from these DOD analyses were used to create a Functional Description of the ISS. Existing AIS software which best satisfied the Functional Description were then converted.

Nine basic applications software modules, comprising approximately 300,000 lines of source code, exist as a result of the conversion. They are CAI Authoring, CAI Presentation, Graphics, Simulation Authoring, Simulation Presentation, CMI Development, CMI Operation, Data Analysis, and Access/Security.

A translator was developed by McDonnell Douglas and Softech to assist in the conversion from CAMIL (Computer-Assisted/Managed Instructional Language), the primary language of implementation for the AIS, to Ada, the primary language of implementation for the ISS. The translator was capable of translating approximately 80% of a CAMIL program to correct Ada. Approximately 20% of a CAMIL program was either partially translated or untranslated. These areas were clearly marked with manual translation hints in the resultant Ada.

The ISS Application Support Environment (ASE) was developed concurrently with conversion of the applications software. The purpose of the ASE is twofold: First, it provides portability to the ISS, given that machine and operating system dependencies are implemented at the lowest level of the support environment; second, it provides a variety of basic runtime support services to ISS applications software to assist in the areas of user interaction, data processing, and storage and retrieval of data.

Finally, an important aspect in the success of the Standardized Software project was a microcomputer analysis, conducted to analyze and recommend small machines capable of executing ISS software modules. An MC68000 basic Pacific Microcomputer PM200 was procured by the Air Force as a result of the study. Key software modules were successfully compiled and executed on the system, demonstrating the concept of ISS transportability and feasibility of running the ISS on a microcomputer.

Accession For
NTIS
By
Dist
Codes
Dist
A-1

QUALITY INSPECTED 3

PREFACE

A number of individuals have contributed significantly in the successful design and implementation of the ISS. Alphabetically, these individuals are: Dick Bolz (AFHRL/ID), Dave Grossart (MDAC), Alan Hallauer (MDAC), Alan Marshall (AFHRL/ID), Glenn McBride (MDAC), Anne Montgomery (MDAC), Dave Pflasterer (MDAC), Steve Schaefer (MDAC), Rick Shelgren (MDAC), Mark Weinberg (MDAC), and Dana Wunderlich (MDAC).

TABLE OF CONTENTS

		Page
1.0	INTRODUCTION	1
2.0	PROJECT DESCRIPTION	1
3.0	MAJOR ACCOMPLISHMENTS	2
3.1	Converted Applications Software	2
3.2	CAMIL-to-Ada Translator	4
3.2.1	Translator Action Routines	5
3.2.2	Translation Routines	6
3.3	The Application Support Environment	7
3.3.1	Terminal Communication	7
3.3.2	Data Management	9
3.3.3	Inter-Process Communication	9
3.3.4	Text Handling	9
3.3.5	Program Control	10
3.3.6	Mathematical Services	11
3.4	Software Portability	11
3.4.1	Ada Programming Language Requirements	11
3.4.2	Host Operating System Requirements	12
3.4.3	Hardware Requirements	12
3.4.3.1	Process/Peripheral Requirements	14
3.4.3.2	Display Station Requirements	14
3.4.4	VAX-11/780 And PM200 Implementations	15
4.0	ISS POTENTIAL	15
4.1	Current Implementations	16
4.2	Future Implementations	16
4.3	The Configurable ISS	16
4.4	ISS Tailoring	16
4.5	The ISS Micro As A Central Processor	17
5.0	CONCLUSIONS AND RECOMMENDATIONS	18
REFERENCES		18

LIST OF FIGURES

Figure		Page
1	ISS Software Structure	3
2	Automatic Translation from CAMIL to Ada	5
3	Translator Functional Decomposition	6
4	ISS Software Structure Illustrating the Divided Application Support Environment	7
5	Inter-Process Communication	10
6	Example of Cost/Performance Alternatives	17

LIST OF TABLES

Table		Page
1	VML Procedures Utilizing Host Operating System Software	13
2	VML Procedures Fulfilling Performance Requirements	14

INSTRUCTIONAL SUPPORT SOFTWARE SYSTEM

1.0 INTRODUCTION

The prototype Advanced Instructional System (AIS) was designed as a research and development test bed for technical training. As such, it demonstrated that individualized computer-assisted instruction (CAI) and computer-managed instruction (CMI) are directly applicable to an operational Air Force training environment.

Although demonstrated as feasible, the original system was not transportable; therefore, exploitation of the training technology was limited. In order to correct this problem, the Technical Training Division of the Air Force Human Resources Laboratory (AFHRL) awarded a contract to the McDonnell Douglas Astronautics Company to create a transportable system. This Standardized Software project had as the major goal to create a transportable instructional system that is implementable on low-cost minicomputers and microcomputers in order to expand into the appropriate DOD training environments. The transportable system, called the Instructional Support Software (ISS) system, has been developed and alpha tested. An operational test of the system is projected to begin during the third quarter of 1985.

The Air Force set forth several key requirements to accomplish its major goal. It was decided that Ada, the newly standardized DOD language, would be used to enhance system portability. Ada is the ideal language in which to implement transportable software, given its mission as a standard high-order language that will become available on many machines.

Another requirement was to develop a set of generalized interfaces to enhance portability. By embedding machine dependencies low into the ISS support software, the necessary interface to the host operating system could be accomplished in a portable way.

The final key requirement was to create application software made up of modular components to support the execution of individual portions of the ISS. Components such as Authoring, Graphics, and CAI Presentation were to be created for execution on an individual basis. By allowing potential DOD customers the capability to choose only a subset of the entire ISS, particular needs can be met at lower cost.

In order to report the significance and results of the Standardized Software project, the following sections of this report will provide a project description, a statement of the major accomplishments of the project, information on ISS potential, and conclusions and recommendations.

2.0 PROJECT DESCRIPTION

Early in the project, McDonnell Douglas teamed with the Denver Research Institute to determine the functional requirements of the ISS. Key DOD training systems were examined to determine the feasibility of adding certain key features to the ISS at reasonable cost. Also, several key DOD training environments were examined to determine what training requirements the ISS would need to address. Existing AIS capabilities, as well as inputs from these DOD analyses, were used to create a functional description of the ISS. Existing AIS software which best satisfied the functional description was then converted.

Nine basic application software modules, comprising approximately 300,000 lines of source code, exist as a result of the conversion. They are CAI Authoring, CAI Presentation, Graphics, Simulation Authoring, Simulation Presentation, CMI Development, CMI Operation, Data Analysis, and Access/Security.

A translator was developed to assist in the conversion from CAMIL (Computer-Assisted/Managed Instructional Language), the primary language of implementation for the AIS, to Ada, the primary language of implementation for the ISS. The translator was capable of translating approximately 80% of a CAMIL program to correct Ada. Approximately 20% of a CAMIL program was either partially translated or untranslated. These areas were clearly marked with manual translation hints in the resultant Ada.

The ISS Application Support Environment (ASE) was developed concurrently with conversion of the application software. The purpose of the ASE is twofold. First, it provides portability to the ISS, given that machine and operating system dependencies are implemented at the lowest level of the support environment. Second, it provides a variety of basic runtime support services to ISS application software to assist in the areas of user interaction, data processing, and storage and retrieval of data.

Finally, an important aspect in the success of the Standardized Software project was a microcomputer analysis, conducted to analyze and recommend small machines capable of executing ISS software modules. An MC68000-based Pacific Microcomputer PM200 was procured by the Air Force as a result of the study. Key software modules were successfully compiled and executed on the system, demonstrating the concept of ISS transportability and the feasibility of running the ISS on a microcomputer.

3.0 MAJOR ACCOMPLISHMENTS

The goals set forth at the beginning of the Standardized Software project have been accomplished. Applications software which best satisfies the functional description has been converted or developed. The developed system is portable. After the software had first been produced on the development machine (VAX-11/780) and then transported to the PM200 microcomputer, the concept of portability had been demonstrated. And finally, the system has been implemented on a low-cost minicomputer and microcomputer.

The ISS is organized using a layered shell approach to allow for maximum ease of maintenance and transportability. The design philosophy of the software is depicted in Figure 1. ISS users interface with a set of programs called the Applications Software (described in Section 3.1). The next layer of software, called the Application Support Environment, performs the interfacing tasks necessary to support the Applications Software. The innermost layer is the Operating System Software. This software provides interface to the computer hardware. The advantage of this layered approach is that changes in basic hardware configuration or operating system software are unlikely to adversely affect the ISS Applications Software. The Applications Software is buffered by a layer of support environment software that performs the interfacing between Applications Software and the operating system and hardware.

The following sections discuss in more detail the major accomplishments during the project.

3.1 Converted Applications Software

With the ISS software modules, a user is able to develop, implement, and evaluate training. Table-driven database programs, called editors, are dominant in the system, thereby allowing a user to easily insert, display, and/or change information in the database. The most important characteristic of the editors is that they allow quick access to the database via menus and prompts. No computer programming skills are necessary for ISS system users.

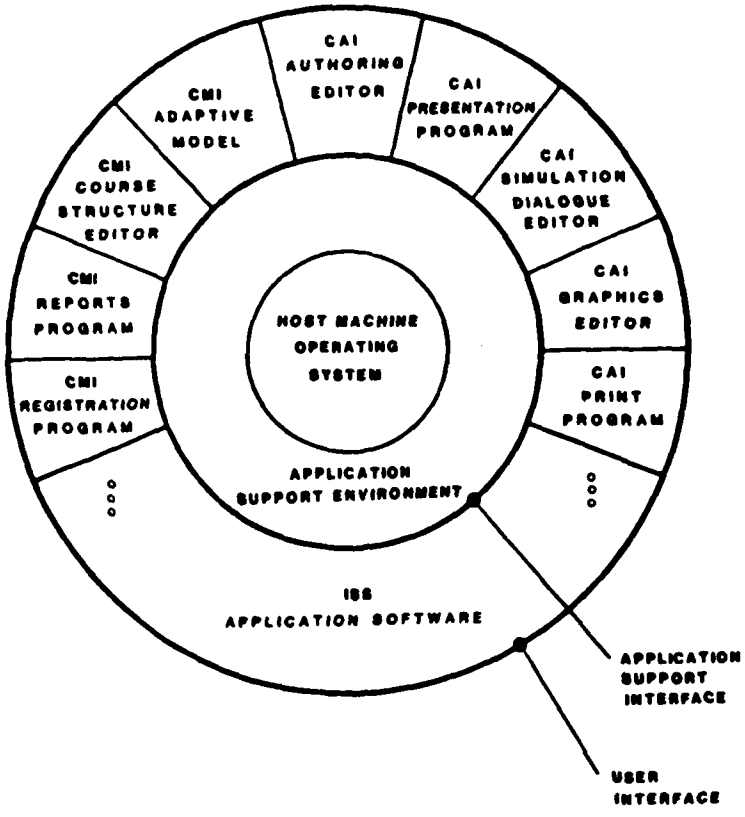


Figure 1. ISS Software Structure.

The applications software is divided into two major functions: the CAI system and the CMI system. The former provides the development and delivery capability for on-line CAI, while the CMI system controls the administrative and management functions for a given installation. Both systems are integrated into the ISS package and share a common base of data and utility programs.

The CAI system allows nonprogrammers to develop and evaluate individualized, interactive CAI modules containing a variety of text and graphics. Using the CAI editors, development of courseware takes the form of an ongoing dialog between the author and the system. There are three major authoring programs in the CAI software: CASS, SID, and GraphEdit. CASS is the screen-frame-oriented, courseware-authoring editor that guides the author via the use of menus. SID is an action-to-screen-oriented simulation courseware authoring editor that also guides the author via menus. GraphEdit is a graphics creation editor for CASS and SID that guides the graphics artist through input from many possible sources such as the keyboard, a digitizer tablet, a touch panel, a joy-disk, or a light pen. Like the other editors, GraphEdit is menu-driven.

CAI lessons are delivered to students through CAI software modules: CAI Presentation (CAIPres) and Simulation Presentation (SIDPres). These modules provide the student with all CAI and simulation lessons and allow for interaction, screen dynamics, branching, remediation, feedback, and prompts. Like the authoring modules, CAIPres and SIDPres are menu-driven and user-friendly.

The CMI system provides comprehensive management information and administration implementation. It controls the scheduling of assignments, testing, remediation, and enrichment activities for each student.

Four major editors assist the curriculum developer in describing the curriculum. By using these four editors, the Curriculum Definition Editor (CDE), the Course Structure Editor (CSE), the Lesson Definition Editor (LDE), and the Test Editor (TESTED), curriculum developers and managers can define increasingly detailed characteristics of a curriculum. After the development task is complete, the CMI operation programs administer the curriculum.

Student registration, course structure management, resource management, and student self-pacing are maintained by three CMI programs comprising the CMI operations module. The programs are Student Logon, Student Registration, and the Adaptive Model.

Evaluation of individual student and overall course progress is monitored and recorded by three CMI evaluation programs comprising the Data Analysis module. The programs are Course Evaluation Summary (CES), Test Item Evaluation (TIE), and the Data Extraction Program (DEP). These programs report data such as the standard deviation/mean times and test scores for a lesson, course, or entire curriculum (CES), determine standard deviation/mean time and scores for specific tests and test items (TIE), and select any available data, ad hoc, to run analysis for curriculum evaluation and student/group performance (DEP).

The Access/Security module is used to define the access a user will have to the ISS. The Access/Security software is necessary to allow operation of all of the other ISS software. The software contains two programs: USRED and LOGON.

Users of a computer-based training system are typically concerned with the security measures that control access to the system. USRED allows system managers to control access to all ISS components and software. It identifies users, database programs, and courses to the ISS system, and authorizes and/or restricts the access of individual users to ISS editors, database programs, and courses. USRED can also regulate a user's level of access within a specific editor, database program, or course. One may, for example, give a user permission to add and/or change information in a specific course, but not to delete information. One may give another user permission to display information but not to manipulate it in any way.

The LOGON program provides all users with access to the ISS. It also functions as a security checkpoint by requiring the user to enter a specific ID followed by a specific password. LOGON compares the information entered by the user with information that has been stored by USRED in the user record. If no match is found, LOGON instructs the user to reenter the data.

Students entering LOGON are automatically transferred to the Student Logon program. Once in Student Logon, a student user can interact with the ISS for training activities.

Non-student users are admitted to the system via LOGON. Here, the user is presented with a list of editors and database programs that have been authorized the user via USRED.

3.2 CAMIL-to-Ada Translator

A translator has been developed to assist in the conversion from CAMIL (AIS-3.8-1674, 1979) to Ada. The translator is capable of translating approximately 80% of a CAMIL program to correct Ada (ANSI/MIL-STD-1815A, 1983). Approximately 20% of a CAMIL program is either partially translated or untranslated. These partially or untranslated areas are clearly marked with manual translation hints in the resultant Ada. Figure 2 illustrates the mechanism used to translate a CAMIL program to Ada and to compile and link that Ada program into executable form.

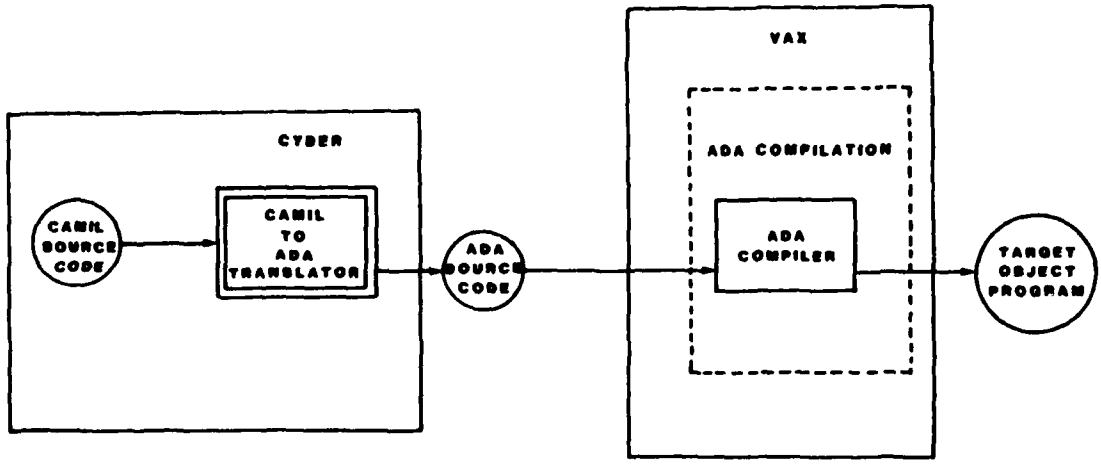


Figure 2. **Automatic Translation from CAMIL to Ada.**

The functional units that comprise the translator are generally the same as those that comprise the CAMIL compiler. The major differences in the translator functional units are as follows: (See Figure 3 and DD1017F019, 1974.)

1. Modification of production action routines.

2. Replacement of code generation routines with Ada source generation routines.

3. Additions to data structures for symbol and type table entries.

Other minor differences exist. For example, the translator version of the lexical analyzer and parser saves information about comments appearing in the source so that they may be preserved in the Ada translation. Figure 3 illustrates the combined top-level functional decomposition of the CAMIL compiler and the translator. Note that the portion labeled Code Generator is part of the CAMIL compiler and is replaced by the Translation Routines and Ada Source Generator in the translator. All the other top-level functional units are shared by both the compiler and the translator (with the differences already mentioned). Sections 3.2.1 and 3.2.2 contain a general description of how a translation is accomplished.

3.2.1 *Translator Action Routines*

Each time the translator's parser recognizes a particular construct or portion of a construct (called production) in the CAMIL language, it invokes an associated action routine. These action routines are responsible for checking semantic restrictions and building data structures (called semantic frames) to represent the construct recognized. Syntax checking is done by the parser.

The semantic frames generated by action routines are placed on a stack which is pushed and popped by the parser as constructs are recognized. In this way, an action routine for a

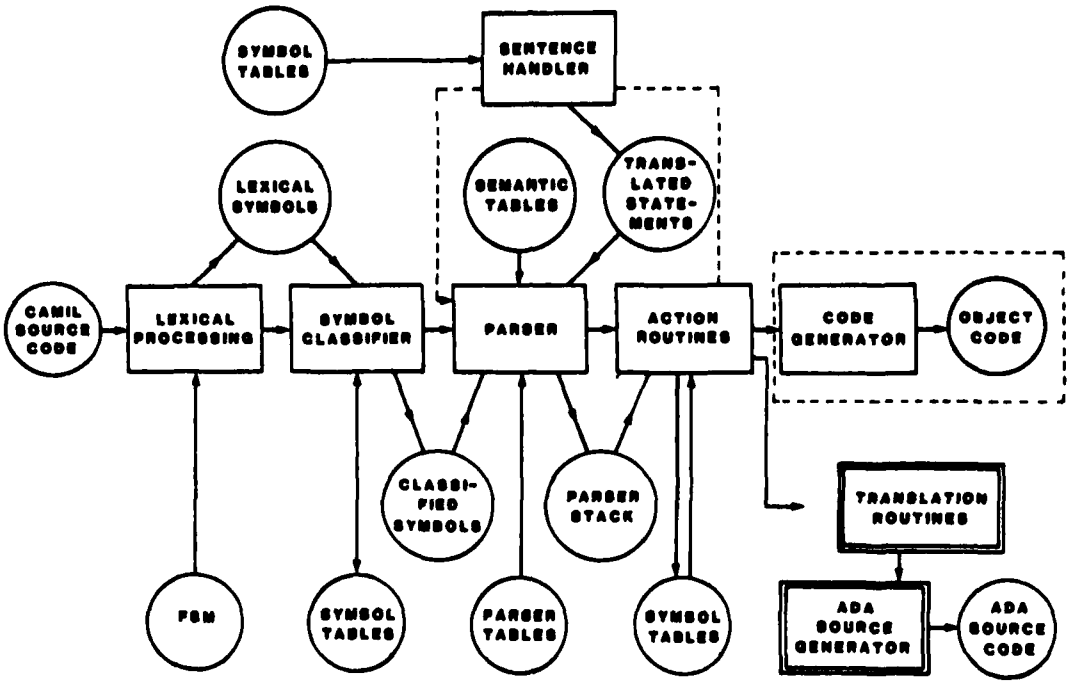


Figure 3. Translator Functional Decomposition.

low-level production (for example, one that processes an operator expression) can pass information about that production to another action routine associated with a higher level production (for example, an assignment statement production in which the operator expression forms the right-hand side).

The action routines representing the declaration productions (e.g., variable declarations) build symbol and type table entries using information that has been saved in semantic frames by action routines.

Similarly, action routines invoked for statement level productions (e.g., assignment) call translation routines to generate code passing them information saved in semantic frames (e.g., variable and expression frames) built by lower frames (on the parser stack), which represent the parts of the right-hand side of the semantic production. The action routine usually generates a new semantic frame that is input to higher level productions; however, some action routines generate a symbol or type table entry and/or call a translation routine to generate Ada source code.

3.2.2 Translation Routines

The translation routines are organized in a manner that roughly parallels the productions. These routines are initially invoked by statement level action routines and internally invoke each other to complete translation of a statement. The transition routines call the routines in the Ada source generator that actually generate the source string. The translation routines operate by "walking" through the semantic tree and calling Ada source generation routines.

3.3 The Application Support Environment

The purpose of the Application Support Environment is twofold. First, it provides portability to the ISS, given that machine and operating system dependencies are implemented at the lowest level of the support environment. (See Section 3.4.) Second, it provides a variety of basic runtime support services to ISS applications software to assist in the areas of user interaction, data processing, and storage and retrieval of data.

The Application Support Environment is divided into two layers: the Application Support Layer (ASL) and the Virtual Machine Layer (VML). Figure 4 illustrates the division of the Application Support Environment in order to support the applications software and to interface to the host operating system. This section of the report describes the support services provided by the ASL to the applications software. The VML is discussed in Section 3.4. The ASL provides software in the areas of terminal communications, data management, inter-process communication, text handling, program control, and mathematical services.

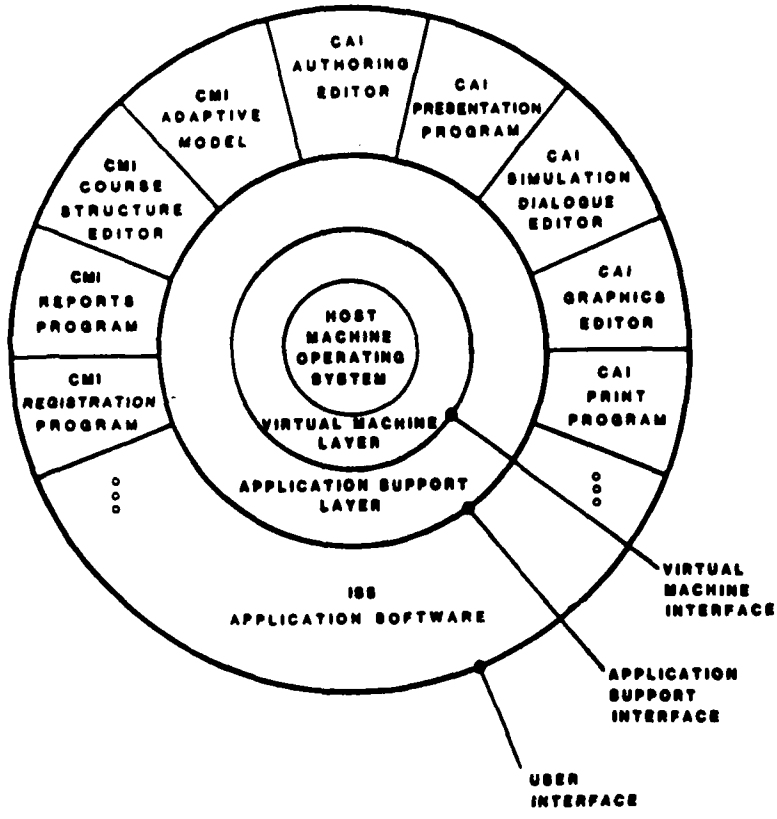


Figure 4. ISS Software Structure Illustrating the Divided Application Support Environment.

3.3.1 Terminal Communication

The Terminal Communication component provides the functional interface between the ISS applications software and the terminal devices used to communicate with applications software users. Terminal devices can be easily added since individual terminal characteristics are stored in data tables. All that is necessary for the ISS to support a new terminal type is to enter the data describing the new terminal into the terminal definition table. The full set of combined

text/graphics display characteristics and capabilities provided by the Terminal Communication component are as follows:

1. Characters per line range from 60 to 128.

2. Lines per screen range from 24 to 64.

3. Lines are numbered from 1 to L with line 1 at the top of the screen and line L at the bottom of the screen.

4. Columns are numbered from 1 to C with column 1 at the leftmost character cell position and column C at the rightmost position.

5. Random cursor positioning is possible to any character position within the character matrix defining the screen.

6. Random cursor positioning to any dot position within the dot matrix is possible.

7. Double height and width characters, as well as normal height and width characters, are displayable.

8. Lines of dots from the current dot position to a different dot position are displayable.

9. Either a complete or partial circle of dots with a given radius is displayable, starting at the current cursor position.

10. Text and graphics are displayable in eight different colors.

11. It is possible to select whether background or foreground elements can blink.

The terminal keyboard is the means by which users of an application program input data to that program. Keypresses are interpreted by the Terminal Communication component and acted upon where appropriate. Provisions are made for four distinct types of keys. These keys, as well as the set of keyboard characteristics and capabilities, are as follows:

1. Textual Data Keys - This set of keys represents the printable character symbols as defined in the ASCII character set.

2. Function Keys - This set of keys has special meaning to the Terminal Communication component. Entry of an enabled function key triggers a preemptive transfer of control from the current point of execution within an application program to a handler area previously declared within the program.

3. Action Keys - This set of keys has special meaning to the Terminal Communication component. Entry of an action key causes the Terminal Communication component to act on the data being assembled by a keyboard read operation (e.g., deletion of a character by pressing the delete key).

4. Terminal Control Keys - This set of keys has special meaning to the Terminal Communication component. Each terminal control key represents a special terminal control function which can be performed by pressing that key. The terminal control functions generally affect the current display screen attributes (e.g., color and blink).

3.3.2 Data Management

The Data Management component consists of the ISS database and the necessary operations required by the applications software for accessing and maintaining the database. The database is the data storage system for all of the data objects used by the ISS applications software. It supports Indexed Sequential, Direct Access, and Sequential files. The names of the files are stored in a directory within the database along with sufficient additional information necessary to access the file. A file cannot span disks but is otherwise not limited in size. The characteristics of the different file types are as follows:

1. Indexed Sequential (ISAM) Files - An ISAM file is capable of containing zero or more records, each of which may contain a variable amount of data. For each ISAM file in the data base, a "key" is defined which designates that a specific portion of each record be used to define a sequential ordering of the records contained in that file. Each record is at least long enough to contain the entire key. An index sufficient to map key values to record locations, in order to support random record accessing, is maintained for each ISAM file. An ISAM key can be a maximum of 127 bytes.

2. Direct Access (DA) Files - A DA file is capable of containing zero or more records, each of which may contain a variable amount of data. Each record within a DA file is associated with a relative record number which defines the sequential ordering of the records contained in that file. An index sufficient to map relative record numbers to record locations, in order to support random record accessing, is maintained for each Direct Access file. The maximum relative record number in use for each file is maintained in order to facilitate the allocation of unused relative record numbers to new records entered into that file.

3. Sequential Files - A sequential file is capable of containing zero or more records, each of which contains a variable amount of data. The records within a sequential file have a sequential ordering based on the order that the records were written into the file.

3.3.3 Inter-Process Communication

The Inter-Process Communication component provides a set of functional capabilities to applications software to enable concurrently active ISS processes to communicate among themselves. Figure 5 depicts that communication. The process is the logical unit of activity within the ISS execution environment and the primary entity relating to Inter-Process Communication. It is an active computing environment that can support the sequential execution of one or more programs. Each active ISS user is associated with a dedicated process and interacts with ISS application programs that execute within this dedicated process. The operating system used by ISS systems and applications software supports the execution of multiple concurrent processes; therefore, multiple, concurrent ISS users are supported.

Each active ISS process is associated with a system-wide ISS Process Index Number which uniquely identifies that process within the system. The Process Index Number is kept across the execution of multiple ISS application programs. At the completion of an ISS process, the Process Index Number is deallocated, allowing other processes to reuse the number.

3.3.4 *Text Handling*

The Text Handling component provides a set of functional capabilities to ISS applications software to manipulate text for display, comparison, assignment, examination, and conversion to/from non-textual data types. Two data types comprise textual data in the ISS: String and

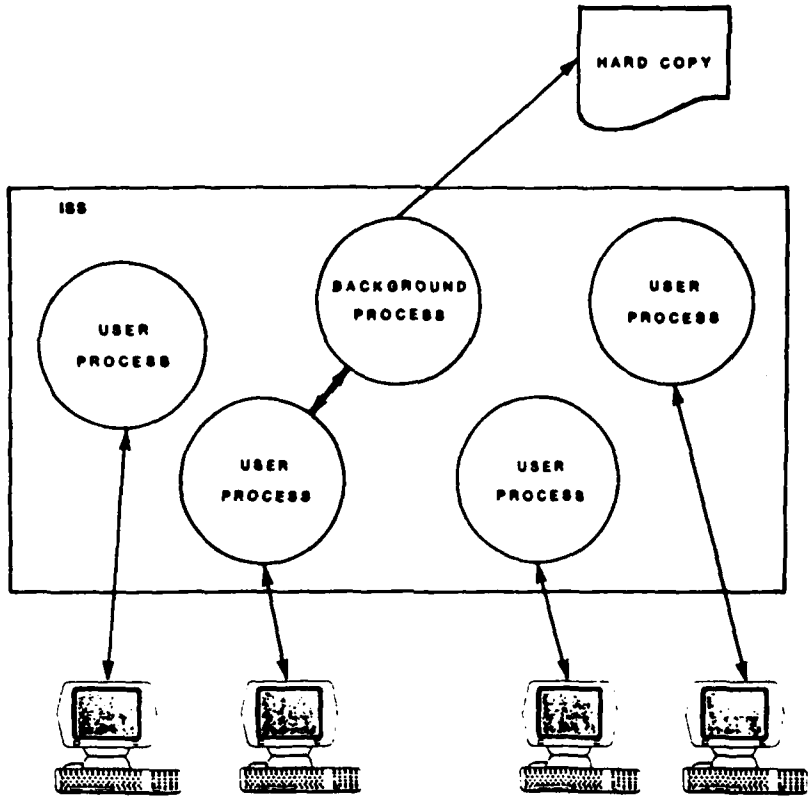


Figure 5. Inter-Process Communication.

Character. String and Character data contain one or more characters of ASCII encoded data representing displayable characters or control characters. String data may also contain generic codes that specify functions to be accomplished by the Terminal Communication component.

A string has an actual length and a maximum length associated with it. A variable string has a termination character to indicate the actual length, with the maximum length indicated at the time of declaration. For the string declaration

```
S : STRING (1..5);
```

the maximum length is 5. By initializing "S" with the assign string procedure

```
ASST(S, "ABC");
```

the actual length is set at 3. "S" would appear as "ABCtc" in computer memory, where tc is the termination character. If the actual length of a variable string is equal to the maximum length, then no termination character is stored in the string. The actual length is determined to be the maximum length when the appropriate string-handling procedures detect no termination character.

3.3.5 Program Control

The Program Control component provides services to assist in the control of the execution flow of ISS programs. The design of existing software for the ISS assumes some additional execution control flow capabilities beyond those available in the ISS implementation language, Ada. The services provided by Program Control are as follows:

1. Program Transfer - An ISS program is able to designate which program should execute within the current process after the current program terminates its execution.

2. Inter-Program Data Passage - An ISS program is capable of passing data for retrieval by the next program specified for execution.

3. Timed Wait - The capability exists for a program to suspend its execution for a designated time interval; the granularity of time is .01 second.

4. Obtaining Date and Time of Day Information - The capability exists for obtaining the current date (year, month, day, and julian date) and current time (hour, minute, and second).

5. Obtaining elapsed and central processing unit (CPU) Time - The capability exists for obtaining the elapsed and CPU time for a session. The variables returned are in 10-millisecond units.

3.3.6 Mathematical Services

The Mathematical Services component provides a set of capabilites to ISS applications software to perform trigonometric, boolean, exponentiation, logarithmic, and other miscellaneous mathematical functions.

Three types comprise the data that can be manipulated by the mathematical functions: INTEGER, FLOAT, and BOOLEAN. The INTEGER and FLOAT data types are implementation defined with respect to magnitude. If a particular host supports 32-bit integers, for example, an ISS INTEGER will be 32 bits. If a host supports 16-bit integers, an ISS INTEGER will be 16 bits. The BOOLEAN data type is represented as an enumeration type in the following way:

```
type boolean is (false, true);
```

3.4 Software Portability

In completing the design and implementation of a transportable system, it has been determined what capabilities are necessary in candidate systems in order to port the ISS to those systems (Marshall, 1983). Requirements of a candidate system can be divided into (a) Ada programming language requirements, (b) host operating system requirements, and (c) hardware requirements (processor/peripherals and terminal). A discussion of those requirements follows in the next three sections of this report, followed by a section describing experience transporting the ISS from the development machine (VAX-11/780) to the PM200 microcomputer.

3.4.1 Ada Programming Language Requirements

Since the ISS is implemented in the Ada language, it will be necessary for any candidate system to provide an Ada compiler. Using such a system does not guarantee successful implementation of the ISS, however. For example, some existing validated compilers impose limitations with respect to code/data sizes and pragma implementation (a pragma conveys information to the compiler but does not affect the correctness of a program). Also, some existing compilers are incomplete implementations and do not provide features needed by the ISS.

If code and/or data are limited to 32k or 64k in a particular implementation of Ada, some ISS programs will not execute without modification on that system.

Certain pragmas are necessary for a production implementation of the ISS (or a means must be devised to equivalently implement the effect of missing pragmas). If the pragma PACK does not sufficiently pack data, data management performance will degrade because records will be much larger than if packing were available. If the pragma INTERFACE is not provided by an Ada compiler, it is awkward to interface to the ISS VML procedures (Figure 4) that are necessary to implement the ISS. If the pragma SUPPRESS is not available, costly execution time checks on subranged integer assignments and array bounds will constantly occur, causing a degradation in performance due to high CPU utilization. Also, while the pragma INLINE is not required, its presence is beneficial since sound design principles can then be used. By appropriately using procedures and functions and declaring them to be compiled inline, performance is not degraded due to costly procedure call and function call linkage.

Note that in certain cases it may be possible to implement a system without the pragma PACK, pragma INTERFACE, and pragma SUPPRESS. Alternate methods causing the same effect should always be considered.

3.4.2 Host Operating System Requirements

In order to fulfill ISS functionality and performance needs, ASL software must utilize VML machine-dependent procedures (Figure 4). These VML procedures, written in a non-Ada language provided by the host system, must be reimplemented on a system to rehost the ISS to that system. The VML procedures are of two different forms: (a) procedures that call host operating system software to gain needed functionality, and (b) procedures that have been written to attain necessary performance. (These procedures are invoked with high frequency but do not use operating system functions; since they are in a machine-dependent, non-Ada language, it can be said that they comprise a portion of host operating system requirements.) In an attempt to minimize the size of the VML, the number of procedures has been kept as small as possible. Tables 1 and 2 describe procedures that are of each form. To clearly depict what host operating system software the ISS requires, the tables specify the VML entry point names and the functionality requirements fulfilled (Table 1) or the performance requirements fulfilled (Table 2). Note that it may be possible to implement some of the VML procedures in Ada on some systems and still meet the functionality and performance requirements. Therefore, in evaluating any future candidate system, a rigid examination should not occur for operating system capabilities that match exactly the requirements given in Tables 1 and 2. Where appropriate, equivalent and acceptable implementations for fulfilling requirements should be considered.

3.4.3 Hardware Requirements

Computer hardware is available in a wide range of varying capacity, functionality, and performance. This section of the report describes the basic requirements of a hardware system capable of developing and executing the ISS.

It is not necessary to require processor, storage, and display station equipment to be packaged either together in a desktop unit or separately in order to successfully develop and execute the ISS on that equipment. For clarity, however, processor/peripheral requirements are described separately from display station requirements.

Table 1. VML Procedures Utilizing Host Operating System Software

VML Entry Point Name	Functionality Requirement Fulfilled
1. BACKSPC	Back space 1 record on a tape file
2. CALL	Call procedure with absolute address
3. CHILDACTIVE	Check for an active sub-process
4. CLOSET	Close a tape file
5. DISMOUNT	Dismount a tape
6. EXP	Raise e to power of input value
7. FCLOSE	Close a file
8. FCREATE	Create a database file
9. FGETS	Read a system file record
10. FOPEN	Open a database file
11. FPUTS	Write a system file record
12. FREAD	Read a database record
13. FREEMEM	Deallocates dynamic storage
14. FREMOVE	Remove a file
15. FSEEK	Position to a database record
16. FWRITE	Write a database record
17. GETT	Read a tape record
18. GET_DATETIME	Return the current date and time
19. GET_TERM_INFO	Return input terminal type
20. GET_TID	Return the terminal identification
21. GET_TIMERS	Return elapsed and CPU time
22. GET_PID	Return the process id
23. IBOOL	Perform specified boolean oper.
24. INTOCHAR	Convert an integer to a character
25. LOW_LOCK	Lock a resource
26. LOW_UNLOCK	Unlock a resource
27. MOUNTT	Mount a tape
28. NEWMEM	Allocate dynamic storage
29. OPENT	Open a tape file
30. PGM_EXISTS	Determine if a program exists
31. PUTT	Put a tape record
32. PRINTFILE	Print a file
33. RAND	Generate a random number
34. RESUMEPROCESS	Resume a process
35. REWINDT	Rewind a tape file
36. RUNPGM	Run a non-background program
37. RUNPROGRAM	Run a background program
38. SHIFT	Perform specified shifting oper.
39. STOPPROCESS	Stop a process
40. SUBMITFILE	Submit a command file
41. SUSPENDPROCESS	Suspend a process
42. TEMPFILE	Create a name for a system file
43. TRANLOG	Translate logical to actual name
44. TRAPMACHINEEXCEPTIONS	Set up to trap machine exceptions
45. TRIG	Perform the specified trig funct.
46. UNIT_READ	Read input from a terminal
47. UNIT_WRITE	Write output to a terminal
48. WAIT	Halt program for input time

Table 2. VML Procedures Fulfilling Performance Requirements

VML Entry Point Name	Performance Requirement Fulfilled
1. COMPAREMEM	Efficiently compares two ranges of memory locations
2. FILLMEM	Efficiently initializes a range of memory locations
3. MOVEMEM	Efficiently moves one range of memory locations to another range of memory locations
4. SEARCHMEN	Efficiently searches a range of memory locations for a specified string of data

3.4.3.1 Process/Peripheral Requirements

Following is a list describing the minimum processor and peripheral requirements for successfully executing the ISS:

1. Processor clock of at minimum 8 MHz.

2. Capability of addressing a minimum of 1 MB of random access memory (RAM) for software development and ISS execution.

3. A minimum of 40-MB hard disk storage for operating system, program development, and ISS applications data storage.

4. A 1-MB floppy disk drive.

Note that it would be possible to implement a more restricted version of the ISS on a system providing less process/peripheral capacities than those listed.

3.4.3.2 Display Station Requirements

Following is a list describing the minimum display-station requirements for successfully presenting ISS displays:

1. Color graphics monitor: An interactive monitor providing both alphanumeric and graphics display capabilities is necessary. Any mixture of text, graphics, and background colors is allowed. Drawing primitives of at least points, vectors, arcs, circles, and rectangles is required. It is necessary for the station to clip picture elements so that screen boundaries are not exceeded. Color and blink attributes must be assignable to any picture primitive.

Following are specific monitor requirements:

a. Screen size of at least 13-inch diagonal.

b. Dot triad spacing of 0.31 mm or better.

c. At least 42-Hz, non-interlaced refresh rate to prevent flicker.

d. Resolution of at least 480 horizontal by 360 vertical.

e. Blink capability.

f. 24 to 32 lines with minimum of 80 character lines.

g. 480+ characters-per-second writing rate.

h. At least 10 microseconds-per-pixel vector-writing rate.

2. Keyboard with function keys and numeric pad.

3. 96 standard ASCII characters plus varying character sizes.

4. If not provided, expandable to support light pen, touch panel, mouse, or other pointing device.

3.4.4 VAX-11/780 And PM200 Implementations

In order to determine the portability of the ISS, software initially implemented on the VAX-11/780 has been implemented on the 68000-based PM200 microcomputer. In general, key ISS modules ported nicely to the PM200 due to (a) the fulfillment, by the PM200, of the Ada programming language requirements, host operating system requirements (UNIX), and hardware requirements discussed in Sections 3.4.1, 3.4.2, and 3.4.3 and (b) the ease with which the VML was reimplemented on the PM200.

The VML consists of approximately 2500 lines of FORTRAN code and 600 lines of Assembly language code on the VAX-11/780. On the PM200, the VML is approximately 1300 lines of code implemented in the C language. By reimplementing this relatively small amount of software to interface to the UNIX operating system and by recompiling the Ada source on the PM200, programs were ported with relative ease.

It should be emphasized that the PM200 implementation was to demonstrate the feasibility of ISS transportability and the execution of the ISS on a microcomputer. Performance and Winchester disk size issues need to be addressed before the PM200 can be considered a production implementation (Section 5.0, Conclusions and Recommendations).

The problems encountered in porting the software were relatively minor. Differences in the command languages for VAX VMS and UNIX had to be reconciled in order to submit programs and print jobs. An open file limit (17) exists in UNIX that does not exist in VMS, and this caused minor modifications to some application programs. And, several compiler bugs were encountered in the Ada compiler on the PM200 (to be expected for early implementations of Ada), causing minor modifications in some of the application programs.

4.0 ISS POTENTIAL

As a result of the Standardized Software project, significant potential uses exist for the ISS: (a) organizations currently using or planning to purchase hardware upon which the ISS now operates can use the system immediately; (b) organizations currently using or planning to purchase a system upon which the ISS can operate will be able to use the ISS after implementation of the Virtual Machine Layer for that system; (c) depending on the training requirements for an organization, the ISS can be delivered, in any combination, as an authoring system, a CAI delivery system, and a CMI system; (d) depending on the training requirements for an organization, the ISS can be tailored to fulfill those requirements; and (e) ISS users can reasonably utilize lower-cost microcomputers, such as the system used in the Standardized Software project, to perform as a central processor. The following sections elaborate on these potential uses.

4.1 Current Implementations

During the project, the ISS was implemented on two systems: the VAX-11/780, using the VMS operating system, and the PM200, using the UNIX operating system. There is significant potential associated with each implementation.

The VMS implementation is significant in that it is available on an ever-broadening and popular series of machines, including the VAX-11/725, VAX-11/730, VAX-11/750, VAX-11/780, and the Micro VAX. DOD organizations currently using or planning to purchase machines from the VAX/VMS line can use the ISS as their training system.

The UNIX operating system also continues to gain in popularity. Unlike VMS in the VAX line, it is implemented on many machines, thereby providing an excellent opportunity for transportation of the ISS technology.

For either implementation, configuration parameters such as central memory size, disk storage space, tape storage space, and terminal type should be carefully considered to best operate the ISS in a particular training environment.

4.2 Future Implementations

In addition to systems utilizing VMS and UNIX, the ISS is implementable on other systems that fulfill the language, operating system, and hardware requirements specified in Sections 3.4.1, 3.4.2, and 3.4.3. The capabilities of a potential system should be examined carefully to determine the feasibility of ISS implementation. For a system fulfilling the requirements, the Virtual Machine Layer must be implemented in order for the ISS to operate successfully on that system.

4.3 The Configurable ISS

Depending on the training requirements for an organization, the ISS can be delivered, in any combination, as a CAI authoring system, a CAI delivery system, and a CMI system. If a training environment requires CAI that has not been developed, a method to systematically and efficiently create courseware is necessary. The software modules comprising the CAI authoring system (CAI Authoring, Graphics, and Simulation Authoring) provide this method. If CAI presentation is required, software modules comprising the CAI delivery system should be used (CAI Presentation and Simulation Presentation). And if management and scheduling of assignments, testing, remediation, and enrichment activities are necessary, support is provided via the CMI system (CMI Development, CMI Operation, and Data Analysis).

4.4 ISS Tailoring

Particular training environments may require hardware devices, terminal types, and/or functional capabilities that are not currently provided in the ISS. With the layered and modular design of the system (Figure 4), new software and device types can be integrated into the ISS with minimal effort. The ISS software is adaptable in nature, with clean interfaces provided by the Ada package specifications. A terminal definition file can be updated to reflect the characteristics of hardware devices to be added to the system.

4.5 The ISS Micro As a Central Processor

By implementing the software on the PM200 microcomputer, it has been shown that the ISS can operate on a more economically feasible machine than minicomputers and mainframes. If, as a hypothetical case, a training organization wanted to support 10 students utilizing 10 curricula, 10 courses, and 10 two-hour CAI lessons, approximately 4 MB of disk storage for instructional data would be required.

The breakdown of the storage requirements is as follows:

1.0 MBytes	Storage for 10 curricula
0.3 MBytes	Storage for 10 courses
0.2 MBytes	Storage for 10 active students
2.5 MBytes	Storage for 10 two-hour lessons
4.0 MBytes	Total storage for instructional data

Also to be considered are ISS program executables which require approximately 15 MB and operating system storage which is approximately 10 MB. The total ISS Winchester storage requirement for this hypothetical case is, therefore, 29 MB. Winchester disk technology is available to easily accommodate this capacity. Additional Winchester space would allow an increase to even more curricula, courses, and student capacity. Current microcomputer technology also allows large amounts of main memory.

With these capabilities currently available in the PM200 and in microcomputer technology in general, a low-cost alternative exists (as compared to minicomputers and mainframes) for certain training applications. Figure 6 depicts an example of the levels of capability from which a training organization could choose, depending on available funds, storage requirements, and computing power necessary. CMI could be performed at the central computer in all cases, and

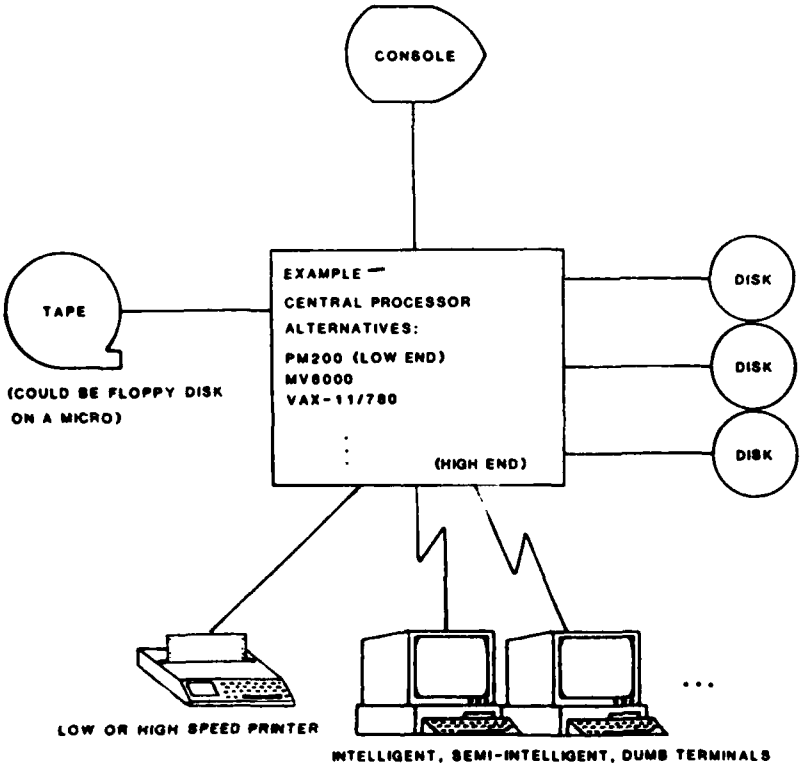


Figure 6. Example of Cost/Performance Alternatives.

depending on the intelligence of the display station, CAI could be performed either under control of the central computer or the display station processor.

5.0 CONCLUSIONS AND RECOMMENDATIONS

The major goals set forth at the beginning of the Standardized Software project have been accomplished. Applications software that best satisfies the Functional Description has been converted or developed. The developed system is portable. Finally, the system has been implemented on a low-cost minicomputer and microcomputer.

A follow-on operational test is recommended for both the VAX and PM200 implementations. During this operational test, significant performance upgrades should be made to the software in order to support the required ISS user load. The test would also allow a user community to evaluate the functionality of the ISS. Appropriate change requests could be issued to AFHRL for evaluation. Enhancements deemed beneficial could then be made in a timely, orderly, and non-disruptive manner.

The PM200 implementation was to demonstrate the feasibility of ISS transportability and execution of the ISS on a microcomputer. While both capabilities have been demonstrated, performance and Winchester disk size issues need to be addressed before the PM200 can be considered a production implementation. Higher-speed, larger-capacity Winchester drives are now available and can be placed in existing slots in the PM200. These are recommended as replacements for the smaller, slower drives used during the Standardized Software project.

Upgrade of the PM200 UNIX system is also recommended to provide more portability.

Finally, consideration should be given to development of a generic data converter in order to transport ISS courseware. With the differences in data packing formats of the many Ada compilers that are and will come into existence, it will be necessary to easily convert those different formats. By developing a generic data converter, courseware portability (as well as software portability) becomes more feasible.

REFERENCES

AIS-3.8-1674. (1979, August). CAMIL reference manual.

ANSI/MIL-STD-1815A. (1983, January). Reference manual for the Ada programming language.

DD1017F019. (1974, January). Critical item development specification for the computer assisted/managed instructional language (CAMIL) component of the advanced instructional system.

Marshall, A.P. (1983, Autumn). Development of a transportable CBI system. Journal of Computer-Based Instruction, 10(3,4), 66-69.

☆U.S. GOVERNMENT PRINTING OFFICE: 1986-659-055-40003

END

FILMED

5-86

DTIC